9 Storing data on hard drives: EBS and instance store

This chapter covers

- Attaching persistent storage volumes to EC2 instance
- Using temporary storage attached to the host system
- Backing up volumes
- Testing and tweaking volume performance
- Differences between persistent (EBS) and temporary volumes (instance store)

Imagine your task is to migrate an enterprise application from being hosted on-premises to AWS. Typically, legacy applications read and write files from a file system. Switching to object storage, as described in the previous chapter, is therefore not possible. Fortunately, AWS offers good old block-level storage as well, allowing you to migrate your legacy application without the need for expensive modifications.

Block-level storage with a disk file system (FAT32, NTFS, ext3, ext4, XFS, and so on) can be used to store files as you would on a personal computer. A *block* is a sequence of bytes, and the smallest addressable unit. The OS is the intermediary between the application that wants to access files and the underlying file system

and block-level storage. The disk file system manages where (at what block address) your files are stored. You can use block-level storage only in combination with an EC2 instance where the OS is running.

The OS provides access to block-level storage via open, write, and read system calls. The simplified flow of a read request goes like this:

1. An application wants to read the file /path/to/file.txt and makes a read system call.
2. The OS forwards the read request to the file system.
3. The file system translates /path/to/file.txt to the block on the disk where the data is stored.

Applications like databases that read or write files by using system calls must have access to block-level storage for persistence. You can't tell a MySQL database to store its files in an object store because MySQL uses system calls to access files.

> **Not all examples are covered by the Free Tier**
>
> The examples in this chapter are *not* all covered by the Free Tier. A special warning message appears when an example incurs costs. Nevertheless, as long as you don't run all other examples longer than a few days, you won't pay anything for them. Keep in mind that this applies only if you created a fresh AWS account for this book and nothing else is going on in your AWS account. Try to complete the chapter within a few days; you'll clean up your account at the end.

AWS provides two kinds of block-level storage:

- A *persistent block-level storage* volume connected via network—This is the best choice for most problems, because it is independent from your virtual machine's life cycle and replicates data among multiple disks automatically to increase durability and availability.
- A *temporary block-level storage* volume physically attached to the host system of the virtual machine—This is interesting if you're optimizing for performance, as it is directly attached to the host system and therefore offers low latency and high throughput when accessing your data.

The next three sections will introduce and compare these two solutions by connecting storage with an EC2 instance, doing performance tests, and exploring how to back up the data.

9.1 Elastic Block Store (EBS): Persistent block-level storage attached over the network

Elastic Block Store (EBS) provides persistent block-level storage with built-in data replication. Typically EBS is used in the following scenarios:

- Operating a relational database system on a virtual machine.
- Running a (legacy) application that requires a filesystem to store data on EC2.
- Storing and booting the operating system of a virtual machine.

An EBS volume is separate from an EC2 instance and connected over the network, as shown in figure 9.1. EBS volumes:

- Aren't part of your EC2 instances; they're attached to your EC2 instance via a network connection. If you terminate your EC2 instance, the EBS volumes remain.
- Are either not attached to an EC2 instance or attached to exactly one EC2 instance at a time.
- Can be used like typical hard disks.
- Replicate your data on multiple disks to prevent data loss due to hardware failures.

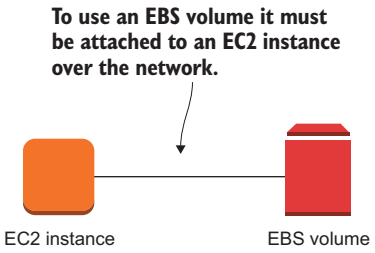


Figure 9.1 EBS volumes are independent resources but can only be used when attached to an EC2 instance.

EBS volumes have one big advantage: they are not part of the EC2 instance; they are an independent resource. No matter if you stop your virtual machine or your virtual machine fails because of a hardware defect, your volume and your data will remain.

WARNING You can't attach the same EBS volume to multiple virtual machines! See chapter 10 if you are looking for a network filesystem.

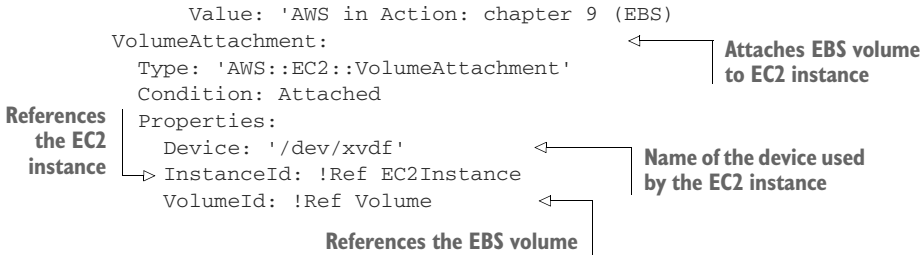
9.1.1 Creating an EBS volume and attaching it to your EC2 instance

Let's return to the example from the beginning of the chapter. You are migrating a legacy application to AWS. The application needs to access a filesystem to store data. As the data contains business-critical information, durability and availability are important. Therefore, you create an EBS volume for persistent block storage. The legacy application runs on a virtual machine, and the volume is attached to the VM to enable access to the block-level storage.

The following bit of code demonstrates how to create an EBS volume and attach it to an EC2 instance with the help of CloudFormation:

```
EC2Instance:                                    <-- Defines the EC2 instance
  Type: 'AWS::EC2::Instance'
  Properties:                                   <-- We are skipping the properties of
    # [...]                                         the EC2 instance in this example.

Volume:                                         <-- Defines the EBS volume
  Type: 'AWS::EC2::Volume'
  Properties:
    AvailabilityZone: !Sub ${EC2Instance.AvailabilityZone}
    Size: 5                                     <-- Creates a volume with 5 GB capacity
    VolumeType: gp2                             <-- Default volume type based on SSD
    Tags:
    - Key: Name
```



An EBS volume is a standalone resource. This means your EBS volume can exist without an EC2 instance, but you need an EC2 instance to access the EBS volume.

9.1.2 Using EBS

To help you explore EBS, we've prepared a CloudFormation template located at http://mng.bz/6q51. Create a stack based on that template by clicking the CloudFormation Quick-Create link at http://mng.bz/5o2D, select the default VPC and a random subnet, and set the `AttachVolume` parameter to `yes`. Don't forget to check the box marked I acknowledge that AWS CloudFormation might create IAM resources. After creating the stack, copy the `PublicName` output and connect via SSH: `ssh -i` *$PathToKey*`/ mykey.pem ec2-user@`*$PublicName*.

You can see the attached EBS volumes using `fdisk`. Usually, EBS volumes can be found somewhere in the range of /dev/xvdf to /dev/xvdp. The root volume (/dev/xvda) is an exception—it's based on the AMI you choose when you launch the EC2 instance, and contains everything needed to boot the instance (your OS files):

```
$ sudo fdisk -l
Disk /dev/xvda: 8589 MB [...]
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: gpt


  #         Start          End     Size  Type            Name
  1          4096     16777182       8G  Linux filesyste Linux
128          2048         4095       1M  BIOS boot parti BIOS Boot Partition

Disk /dev/xvdf: 5368 MB [...]
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

The root volume, an EBS volume with a size of 8 GiB.

An additional volume, an EBS volume with a size of 5 GiB.

The first time you use a newly created EBS volume, you must create a filesystem. You could also create partitions, but in this case the volume size is only 5 GB, so you probably don't want to split it up further. As you can create EBS volumes in any size and attach multiple volumes to your VM, partitioning a single EBS volume is uncommon.

Instead, you should create volumes at the size you need; if you need two separate scopes, create two volumes. In Linux, you can create a filesystem on the additional volume with the help of `mkfs`. The following example creates an ext4 file system:

```
$ sudo mkfs -t ext4 /dev/xvdf
mke2fs 1.42.12 (29-Aug-2014)
Creating filesystem with 1310720 4k blocks and 327680 inodes
Filesystem UUID: e9c74e8b-6e10-4243-9756-047ceaf22abc
Superblock backups stored on blocks:
  32768, 98304, 163840, 229376, 294912, 819200, 884736

Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done
```

After the filesystem has been created, you can mount the device:

```
$ sudo mkdir /mnt/volume/
$ sudo mount /dev/xvdf /mnt/volume/
```

To see mounted volumes, use `df -h`:

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        489M   60K  488M   1% /dev
tmpfs           497M     0  497M   0% /dev/shm
/dev/xvda1      7.8G  980M  6.7G  13% /           <-- Root volume
/dev/xvdf       4.8G   10M  4.6G   1% /mnt/volume  <-- Additional volume
```

EBS volumes are independent from your virtual machine. To see this in action, you will save a file to a volume, unmount, and detach the volume. Afterward, you will attach and mount the volume again. The data will still be available!

```
$ sudo touch /mnt/volume/testfile    <-- Creates testfile in /mnt/volume/
$ sudo umount /mnt/volume/           <-- Unmounts the volume
```

Update the CloudFormation stack, and change the `AttachVolume` parameter to `no`. This will detach the EBS volume from the EC2 instance. After the update is completed, only your root device is left:

```
$ sudo fdisk -l
Disk /dev/xvda: 8589 MB, 8589934592 bytes, 16777216 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

```
Disk label type: gpt


  #        Start          End    Size  Type            Name
  1         4096     16777182      8G  Linux filesyste Linux
128         2048         4095      1M  BIOS boot parti BIOS Boot Partition
```

The testfile in /mnt/volume/ is also gone:

```
$ ls /mnt/volume/testfile
ls: cannot access /mnt/volume/testfile: No such file or directory
```

Now you'll attach the EBS volume again. Update the CloudFormation stack, and change the `AttachVolume` parameter to `yes`. After the update is completed, /dev/xvdf is available again:

```
$ sudo mount /dev/xvdf /mnt/volume/
$ ls /mnt/volume/testfile
/mnt/volume/testfile
```

Mounts the attached volume again

Checks whether testfile is still in /mnt/volume/

Voilà: the file test file that you created in /mnt/volume/ is still there.

9.1.3 Tweaking performance

Performance testing of hard disks is divided into read and write tests. To test the performance of your volumes, you will use a simple tool named `dd`, which can perform block-level reads and writes between a source `if=/path/to/source` and a destination `of=/path/to/destination`:

Writes 1 MB 1,024 times

```
$ sudo dd if=/dev/zero of=/mnt/volume/tempfile bs=1M count=1024 \
➥ conv=fdatasync,notrunc
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 16.9858 s, 63.2 MB/s
```

63.2 MB/s write performance

```
$ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
```

Flushes caches

```
$ sudo dd if=/mnt/volume/tempfile of=/dev/null bs=1M count=1024
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 16.3157 s, 65.8 MB/s
```

Reads 1 MB 1,024 times

65.8 MB/s read performance

Keep in mind that performance can be different depending on your actual workload. This example assumes that the file size is 1 MB. If you're hosting websites, you'll most likely deal with lots of small files instead.

But EBS performance is more complicated. Performance depends on the type of EC2 instance as well as the EBS volume type. Table 9.1 gives an overview of EC2 instance types that are EBS-optimized by default as well as types that can be optimized for an additional hourly charge. EC2 instances with EBS optimization benefit by having dedicated bandwidth to their EBS volumes. Input/output operations per second (IOPS) are measured using a standard 16 KB I/O operation size. Performance depends heavily on your workload: read versus write, as well as the size of your I/O operations (a bigger operation size equates to more throughput). These numbers are illustrations, and your mileage may vary.

Table 9.1 What performance can be expected from EBS optimized instance types?

Use case	Instance type	Max bandwidth (MiB/s)	Max IOPS	EBS optimized by default?
General purpose (3rd generation)	m3.xlarge–m3.2xlarge	60-119	4,000-8,000	No
General purpose (4th generation)	m4.large–m4.16xlarge	54–1192	3,600–65,000	Yes
General purpose (5th generation)	m5.large–m5.24xlarge	57–1192	3,600–65,000	Yes
Compute optimized (3rd generation)	c3.xlarge–c3.4xlarge	60–238	4,000–16,000	No
Compute optimized (4th generation)	c4.large–c4.8xlarge	60–477	4,000–32,000	Yes
Compute optimized (5th generation)	c5.large–c5.18xlarge	63–1073	4,000–64,000	Yes
Memory optimized	r4.large–r4.16xlarge	51–1,669	3,000–75,000	Yes
Storage optimized	i3.large–i3.16xlarge	51–1,669	3,000–65,000	Yes
Storage optimized	d2.xlarge–d2.8xlarge	89–477	6,000–32,000	Yes

Depending on your storage workload, you must choose an EC2 instance that can deliver the bandwidth you require. Additionally, your EBS volume must be balanced with the amount of bandwidth. Table 9.2 shows the different EBS volume types and how they perform.

Table 9.2 How EBS volume types differ

Volume type	Volume size	MiB/s	IOPS	Performance burst	Price
General Purpose SSD (gp2)	1 GiB–16 TiB	160	3 per GiB (up to 10,000)	3,000 IOPS	$ $ $
Provisioned IOPS SSD (io1)	4 GiB–16 TiB	500	As much as you provision (up to 50 IOPS per GiB or 32,000 IOPS)	n/a	$ $ $ $

Table 9.2 How EBS volume types differ *(continued)*

Volume type	Volume size	MiB/s	IOPS	Performance burst	Price
Throughput Optimized HDD (st1)	500 GiB-16 TiB	40 per TiB (up to 500)	500	250 MiB/s per TiB (up to 500 MiB/s)	$ $
Cold HDD (sc1)	500 GiB-16 TiB	12 per TiB (up to 250)	250	80 MiB/s per TiB (up to 250 MiB/s)	$
EBS Magnetic HDD (standard)	1 GiB–1 TiB	40–90	40-200 (100 on average)	Hundreds	$ $

Here are typical scenarios for the different volume types:

- Use *General Purpose SSD (gp2)* as the default for most workloads with medium load and a random access pattern. For example, use this as the boot volume or for all kinds of applications with low-to-medium I/O load.
- I/O-intensive workloads access small amounts of data randomly. *Provisioned IOPS SSD (io1)* offers throughput guarantees, for example, for large and business-critical database workloads.
- Use *Throughput Optimized HDD (st1)* for workloads with sequential I/O and huge amounts of data, such as Big Data workloads. Don't use this volume type for workloads in need of small and random I/O.
- *Cold HDD (sc1)* is a good fit when you are looking for a low-cost storage option for data you need to access infrequently and sequentially. Don't use this volume type for workloads in need of small and random I/O.
- *EBS Magnetic HDD (standard)* is an older volume type from a previous generation. It might be a good option when you need to access your data infrequently.

Gibibyte (GiB) and Tebibyte (TiB)

The terms *gibibyte* (GiB) and *tebibyte* (TiB) aren't used often; you're probably more familiar with gigabyte and terabyte. But AWS uses them in some places. Here's what they mean:

- 1 TiB = 2^{40} bytes = 1,099,511,627,776 bytes
- 1 TiB is ~ 1.0995 TB
- 1 TB = 10^{12} bytes = 1,000,000,000,000 bytes
- 1 GiB = 2^{30} bytes = 1,073,741,824 bytes
- 1 GiB is ~ 1.074 GB
- 1 GB = 10^{9} bytes = 1,000,000,000 bytes

EBS volumes are charged based on the size of the volume, no matter how much data you store in the volume. If you provision a 100 GiB volume, you pay for 100 GiB even if you have no data on the volume. If you use EBS Magnetic HDD (standard) volumes, you must also pay for every I/O operation you perform. Provisioned IOPS SSD (io1)

volumes are additionally charged based on the provisioned IOPS. Use the AWS Simple Monthly Calculator at http://aws.amazon.com/calculator to determine how much your storage setup will cost.

We advise you to use general-purpose (SSD) volumes as the default. If your workload requires more IOPS, then go with provisioned IOPS (SSD). You can attach multiple EBS volumes to a single EC2 instance to increase overall capacity or for additional performance.

9.1.4 Backing up your data with EBS snapshots

EBS volumes replicate data on multiple disks automatically and are designed for an annual failure rate (AFR) of 0.1% and 0.2%. This means on average you should expect to lose 0.5–1 of 500 volumes per year. To plan for an unlikely (but possible) failure of an EBS volume, or more likely a human failure, you should create backups of your volumes regularly. Fortunately, EBS offers an optimized, easy-to-use way to back up EBS volumes with EBS snapshots. A *snapshot* is a block-level incremental backup that is stored in S3. If your volume is 5 GiB in size and you use 1 GiB of data, your first snapshot will be around 1 GiB in size. After the first snapshot is created, only the changes will be saved to S3, to reduce the size of the backup. EBS snapshots are charged based on how many gigabytes you use.

You'll now create a snapshot using the CLI. Before you can do so, you need to know the EBS volume ID. You can find it as the `VolumeId` output of the CloudFormation stack, or by running the following:

```
$ aws ec2 describe-volumes --region us-east-1 \
➥ --filters "Name=size,Values=5"  --query "Volumes[].VolumeId" \
➥ --output text
vol-0317799d61736fc5f                ⟵ The output shows the $VolumeId.
```

With the volume ID, you can go on to create a snapshot:

```
$ aws ec2 create-snapshot --region us-east-1 --volume-id $VolumeId      ⟵ Replace with your $VolumeId.
{
  "Description": "",
  "Encrypted": false,
  "VolumeId": "vol-0317799d61736fc5f",
  "State": "pending",                ⟵ Status of your snapshot
  "VolumeSize": 5,
  "StartTime": "2017-09-28T09:00:14.000Z",
  "Progress": "",
  "OwnerId": "486555357186",
  "SnapshotId": "snap-0070dc0a3ac47e21f"   ⟵ Your $SnapshotId
}
```

Creating a snapshot can take some time, depending on how big your volume is and how many blocks have changed since the last backup. You can see the status of the snapshot by running the following:

```
$ aws ec2 describe-snapshots --region us-east-1 --snapshot-ids $SnapshotId
{
  "Snapshots": [
    {
      "Description": "",
      "Encrypted": false,
      "VolumeId": "vol-0317799d61736fc5f",
      "State": "completed",
      "VolumeSize": 5,
      "StartTime": "2017-09-28T09:00:14.000Z",
      "Progress": "100%",
      "OwnerId": "486555357186",
      "SnapshotId": "snap-0070dc0a3ac47e21f"
    }
  ]
}
```

Replace with your $SnapshotId.

The snapshot has reached the state completed.

Progress of your snapshot

Creating a snapshot of an attached, mounted volume is possible, but can cause problems with writes that aren't flushed to disk. You should either detach the volume from your instance or stop the instance first. If you absolutely must create a snapshot while the volume is in use, you can do so safely as follows:

1. Freeze all writes by running `fsfreeze -f /mnt/volume/` on the virtual machine.
2. Create a snapshot and wait until it reaches the pending state.
3. Unfreeze to resume writes by running `fsfreeze -u /mnt/volume/` on the virtual machine.
4. Wait until the snapshot is completed.

Unfreeze the volume as soon as the snapshot reaches the state *pending*. You don't have to wait until the snapshot has finished.

With an EBS snapshot, you don't have to worry about losing data due to a failed EBS volume or human failure. You are able to restore your data from your EBS snapshot.

To restore a snapshot, you must create a new EBS volume based on that snapshot. Execute the following command in your terminal, replacing *$SnapshotId* with the ID of your snapshot.

Adds regularization parameters

```
$ aws ec2 create-volume --region us-east-1 \
➥ --snapshot-id $SnapshotId \
➥ --availability-zone us-east-1a
{
    "AvailabilityZone": "us-east-1a",
    "Encrypted": false,
    "VolumeType": "standard",
    "VolumeId": "vol-0a1afe956678f5f36",
    "State": "creating",
    "SnapshotId": "snap-0dcadf095a785e0bc",
    "CreateTime": "2017-12-07T12:46:13.000Z",
    "Size": 5
}
```

The ID of your snapshot used to create the volume

The $RestoreVolumeId of the volume restored from your snapshot

When you launch an EC2 instance from an AMI, AWS creates a new EBS volume (root volume) based on a snapshot (an AMI includes an EBS snapshot).

Cleaning up AWS OpsWorks

Don't forget to delete the snapshot, the volume, and your stack. The following code will delete the snapshot and volume:

```
$ aws ec2 delete-snapshot --region us-east-1 \
➥ --snapshot-id $SnapshotId
$ aws ec2 delete-volume --region us-east-1 \
➥ --volume-id $RestoreVolumeId
```

Also delete your stack after you finish this section, to clean up all used resources. You created the stack using the UI, so use the UI to delete it. Otherwise, you'll likely be charged for the resources you use.

9.2 Instance store: Temporary block-level storage

An *instance store* provides block-level storage directly attached to the machine hosting your VM. Figure 9.2 shows that the instance store is part of an EC2 instance and available only if your instance is running; it won't persist your data if you stop or terminate the instance. You don't pay separately for an instance store; instance store charges are included in the EC2 instance price.

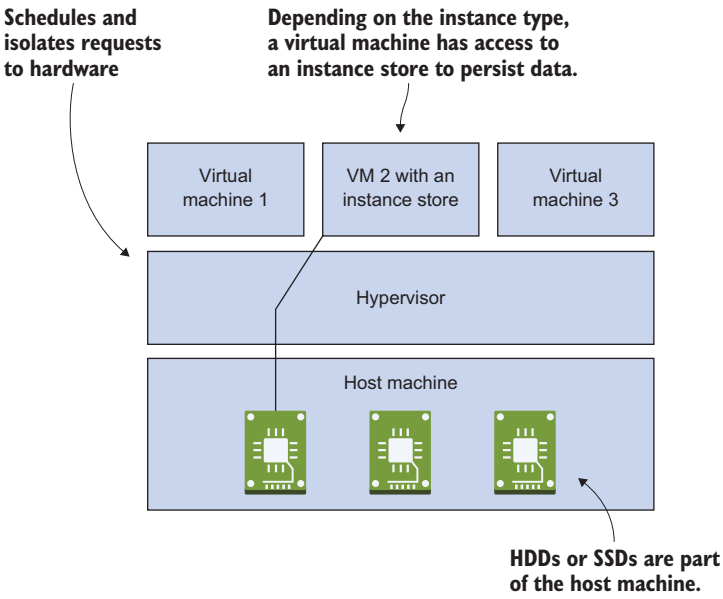


Figure 9.2 The instance store is part of your EC2 instance and uses the host machine's HDDs or SSDs.

In comparison to an EBS volume, which is connected to your VM over the network, the instance store depends upon the VM and can't exist without it. So the instance store will be deleted when you stop or terminate the VM.

Don't use an instance store for data that must not be lost; use it for caching, temporary processing, or applications that replicate data to several servers, as some databases do. If you want to set up your favorite NoSQL database, chances are high that data replication is handled by the application and you can use an instance store.

WARNING If you stop or terminate your EC2 instance, the instance store is lost. *Lost* means all data is destroyed and can't be restored!

AWS offers SSD and HDD instance stores from 4 GB up to 48 TB. Table 9.3 shows a few EC2 instance families providing instance stores.

Table 9.3 Instance families with instance stores

Use case	Instance type	Instance store type	Instance store size in GB
General purpose	m3.medium–m3.2xlarge	SSD	(1 × 4)–(2 × 80)
Compute optimized	c3.large–c3.8xlarge	SSD	(2 × 16)–(2 × 320)
Memory optimized	r3.large–r3.8xlarge	SSD	(1 × 32)–(2 × 320)
Storage optimized	i3.large–i3.16xlarge	SSD	(1 × 950)–(8 × 1,900)
Storage optimized	d2.xlarge–d2.8xlarge	HDD	(3 × 2,000)–(24 × 2,000)

WARNING Starting a virtual machine with instance type m3.medium will incur charges. See https://aws.amazon.com/ec2/pricing/on-demand/ if you want to find out the current hourly price.

To launch an EC2 instance with an instance store manually, open the Management Console and start the Launch Instance wizard as you did in section 3.1. Choose AMI, choose the instance type (m3.medium), and configure the instance details shown in figure 9.3:

1. Click the Add New Volume button.
2. Select instance store 0.
3. Set the device name to dev/sdb.

Tag the instance, configure a security group, and review the stance launch. The instance store can now be used by your EC2 instance.

Listing 9.1 demonstrates how to use an instance store with CloudFormation. If you launch an EC2 instance from an EBS-backed root volume (which is the default), you must define a `BlockDeviceMappings` to map EBS and instance store volumes to device names. Instance stores aren't standalone resources like EBS volumes; the instance store is part of your EC2 instance. Depending on the instance type, you'll have zero, one, or multiple instance store volumes for mapping.

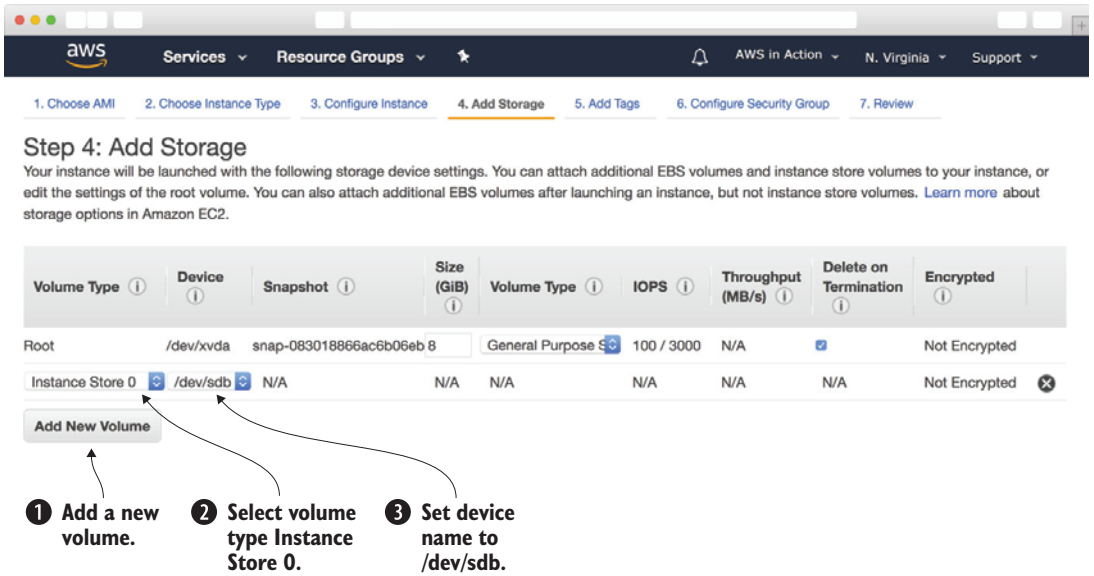


Figure 9.3 Adding an instance store volume manually

Listing 9.1 Connecting an instance store with an EC2 instance with CloudFormation

```
EC2Instance:
  Type: AWS::EC2::Instance
  Properties:
    # [...]
    InstanceType: 'm3.medium'
    BlockDeviceMappings:
    - DeviceName: '/dev/xvda'
      Ebs:
        VolumeSize: '8'
        VolumeType: gp2
    - DeviceName: '/dev/xvdb'
      VirtualName: ephemeral0
```

Choose an instance type with an instance store.

EBS root volume (your OS lives here)

The instance store device will appear as /dev/xvdb.

The instance store is a virtual name like ephemera10 or ephemerall.

Windows-based EC2 instances

The same `BlockDeviceMappings` applies to Windows-based EC2 instances, so the `DeviceName` isn't the same as the drive letter (C:/, D:/, and so on). To go from `DeviceName` to the drive letter, the volume must be mounted. When using Windows, listing 9.1 will still work but the instance store will be available as drive letter Z:/.

Read on to see how mounting works on Linux.

Cleaning up

Don't forget to delete your manually started EC2 instance after you finish this section, to clean up all used resources. Otherwise you'll likely be charged for the resources you use.

9.2.1 Using an instance store

To help you explore instance stores, we created the CloudFormation template located at http://mng.bz/Zu54. Create a CloudFormation stack based on the template by clicking the CloudFormation Quick-Create link at http://mng.bz/V64s.

> **WARNING** Starting a virtual machine with instance type m3.medium will incur charges. See https://aws.amazon.com/ec2/pricing/on-demand/ if you want to find out the current hourly price.

Create a stack based on that template, and select the default VPC and a random subnet. After creating the stack, copy the `PublicName` output to your clipboard and connect via SSH. Usually, instance stores are found at /dev/xvdb to /dev/xvde as shown in the following example:

```
$ sudo fdisk -l
Disk /dev/xvda: 8589 MB [...]
Units = Sektoren of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: gpt


  #         Start          End     Size  Type            Name
  1          4096     16777182       8G  Linux filesyste Linux
128          2048         4095       1M  BIOS boot parti BIOS Boot Partition

Disk /dev/xvdb: 4289 MB [...]
Units = Sektoren of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

EBS device used as root volume containing the OS

The instance store device

To see the mounted volumes, use this command:

```
$ df -h
Filesystem  Size  Used Avail Use% Mounted on
/dev/xvda1  7.8G  1.1G  6.6G  14% /
devtmpfs    1.9G   60K  1.9G   1% /dev
tmpfs       1.9G     0  1.9G   0% /dev/shm
/dev/xvdb   3.9G  1.1G  2.7G  28% /media/ephemeral0
```

The root volume contains the OS.

The instance store volume is mounted automatically.

Your instance store volume is mounted automatically to /media/ephemeral0. If your EC2 instance has more than one instance store volume, ephemeral1, ephemeral2, and so on will be used. Now it's time to run some performance tests to compare the performance of an instance store volume to an EBS volume.

9.2.2 Testing performance

Let's take the same performance measurements as we took in section 9.1.3 to see the difference between the instance store and EBS volumes:

```
$ sudo dd if=/dev/zero of=/media/ephemeral0/tempfile bs=1M count=1024 \
➥ conv=fdatasync,notrunc
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 2.49478 s, 430 MB/s

$ echo 3 | sudo tee /proc/sys/vm/drop_caches
3

$ sudo dd if=/media/ephemeral0/tempfile of=/dev/null bs=1M count=1024
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 0.273889 s, 3.9 GB/s
```

6 × write performance compared with EBS from section 9.1.3

60 × read performance compared with EBS from section 9.1.3

Keep in mind that performance can vary, depending on your actual workload. This example assumes a file size of 1 MB. If you're hosting websites, you'll most likely deal with lots of small files instead. The performance characteristics shows that the instance store is running on the same hardware the virtual machine is running on. The volumes are not connected to the virtual machine over the network as with EBS volumes.

Cleaning up

Don't forget to delete your stacks after you finish this section, to clean up all used resources. Otherwise you'll likely be charged for the resources you use.

9.2.3 Backing up your data

There is no built-in backup mechanism for instance store volumes. Based on what you learned in section 8.2, you can use a combination of scheduled jobs and S3 to back up your data periodically:

```
$ aws s3 sync /path/to/data s3://$YourCompany-backup/instancestore-backup
```

But if you need to back up data, you should probably use more durable, block-level storage like EBS. An instance store is better used for ephemeral persistence requirements.

You will learn about another option to store your data in the next chapter: a network file system.

Summary

- Block-level storage can only be used in combination with an EC2 instance because the OS is needed to provide access to the block-level storage (including partitions, file systems, and read/write system calls).
- EBS volumes are connected to a single EC2 instance via network. Depending on your instance type, this network connection can use more or less bandwidth.
- EBS snapshots are a powerful way to back up your EBS volumes to S3 because they use a block-level, incremental approach.
- An instance store is part of a single EC2 instance, and it's fast and cheap. But all your data will be lost if the EC2 instance is stopped or terminated.